CSE 586 – Distributed Systems
Kafka GitHub Notification System
Project 2
Documentation

Team number: 78
Siddharth Sankaran (50421657)
Shriram Ravi (50419944)

10 December 2021

1. Overview

This purpose of this document is to provide a detailed understanding of the publish-subscribe distributed system that has been implemented as part of Project-2 of the Distributed Systems course. This document will talk about the end-to-end workflow of the system and brief overview of the steps to implement the code.

The system implemented here is a GitHub Notification System that allows a user to receive notifications from their preferred repositories. The user subscribes to different repositories available in GitHub by specifying the owner, repository, etc. in the front-end web application. The user then receives notifications of the latest commits in that repository.

The system also allows the publisher to send advertisements to the users through a simple user interface. Hence every time a user logs in, they can see the advertisements published by the publisher. The publisher can de-advertise to stop the publishing of the advertisement.

Users can freely subscribe and unsubscribe from repositories from time to time making the system dynamic and intuitive.

2. Design

The system is implemented with 3 components:

2.1. Producer (Publisher)

The publishers in this implementation are the different repositories in GitHub that the user has can subscribe to. Every commit to the subscribed repository will the the messages that the

producers send. It will be retrieved through the producer script and pushed to the kafka broker. The publisher end is implemented through a Python Flask application. The GitHub public API is called with specific arguments for owner, repo, time to receive the latest commit messages from the repo since the specified time. The publisher pushes this information (using the send() function) to the kafka broker network which stores the messages in the form of topics. The publisher is dockerized and run as a separate container that performs periodic API calls and pushes information to the broker network.

2.2. Consumer (Subscriber)

The subscribers in this implementation will log into the web application with a username and can subscribe to any public repository available in GitHub. Certain repositories are already given for starters, but new repositories can be added by the user as and when required. This is enabled through a simple front end Flask application that allows the user to pick the repository they want and add a subscription. The consumer application then polls the kafka broker for the subscribed repos (topics) and once a message is received, they are displayed in the notifications tab in the UI.

Based on the list of subscriptions of a user, the front end periodically polls the broker. The front end also has tabs to add new subscriptions and unsubscribe from existing topics.

2.3. Kafka Broker network

The middleware is implemented through a distributed approach with the help of a kafka broker network. The system uses three kafka brokers which are managed through a zookeeper component.

In this implementation, the kafka broker is the intermediary that is responsible for enabling indirect communication between the publisher and the subscriber. Each broker receives messages from different topics from the producer (publisher) end and stores the information in appropriately named topics. The kafka broker network has a very useful property of persisting this data, allowing the consumer to pull the data whenever required. This negates the need for a separate database to store all the messages.

New commits are periodically sent to the broker network and are added to the respective topics sequentially.

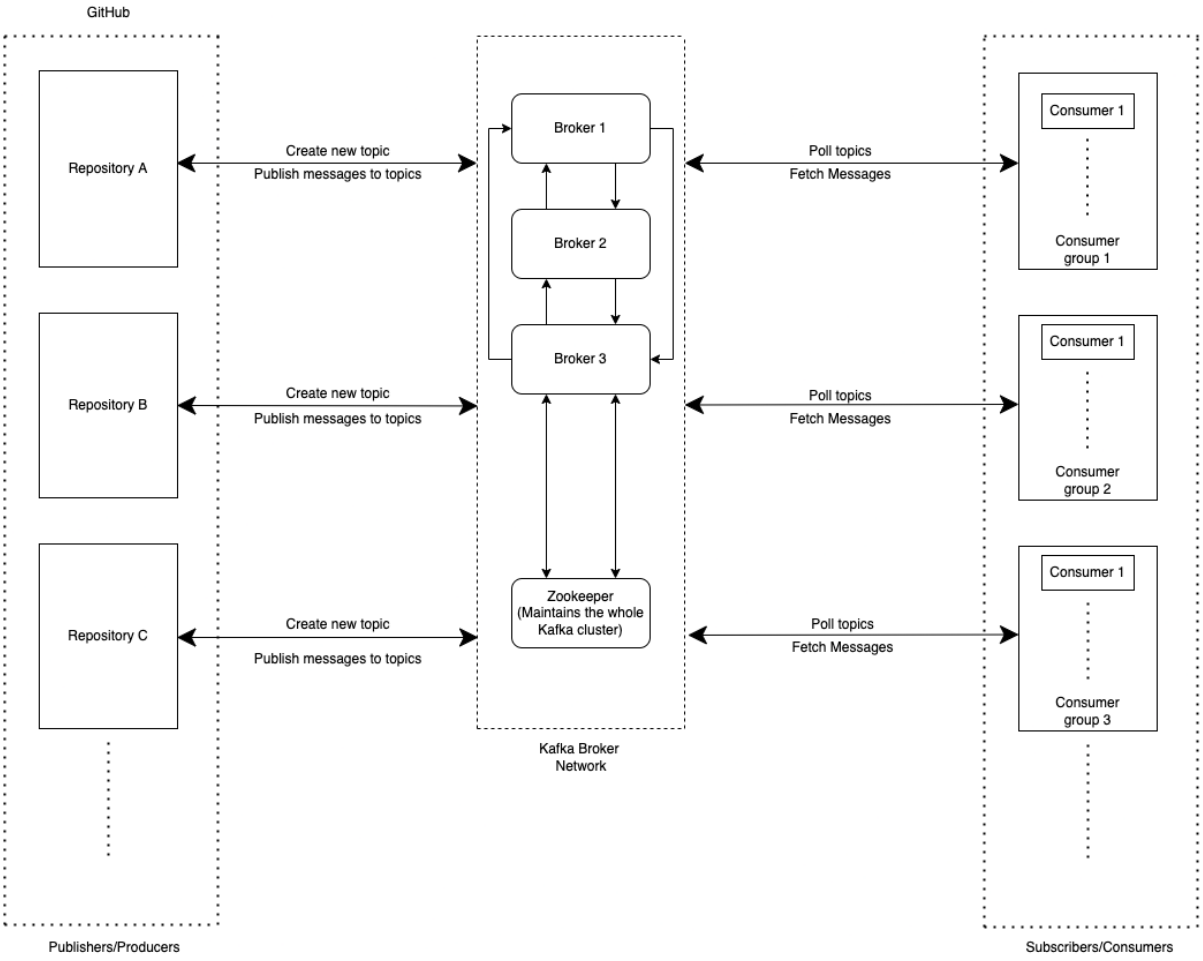
Each consumer will receive messages only meant for that consumer. This is enabled using consumer groups. Each consumer has a separate consumer group that is unique, which is created using the username of the consumer. In this way the system ensures that the offsets of multiple consumers do not get mixed up.

2.3.1. Topics

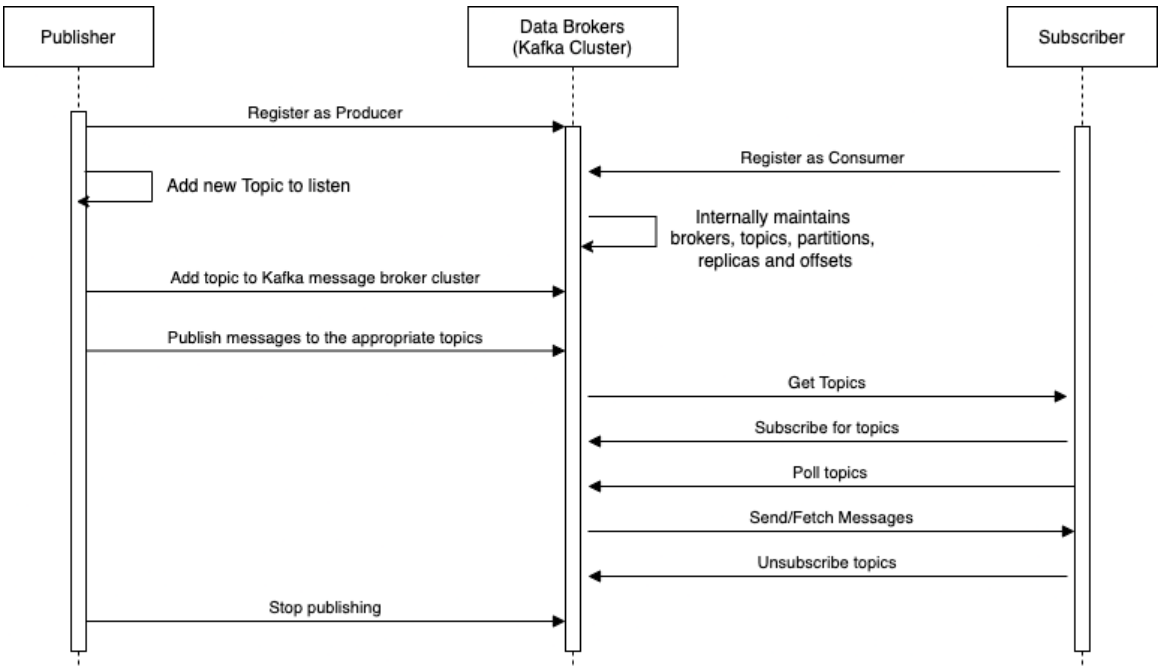
The kafka network uses a combination of topics and partition to store the data sent by the producer. Topic name is the unique identifier used to identify a single repository. It is a created using a combination of platform (GitHub), owner of repo (Mozilla) and repo name (DeepSpeech).

When the application is started there are 5 topics that are already present. More topics can be advertised and published through when required.

3. Architecture Model



4. **Sequence Diagram**



5. **Dependencies and requirements**

For this project we have used a combination of html, CSS, python (Flask), Kafka for implementation

a. Frontend – HTML, CSS, Python (Flask)
b. Backend – Python Flask
c. Broker network – Kafka (Implemented using docker-compose with *wurstmeister/zookeeper, wurstmeister/kafka* images)

6. **Interaction with the application**

- Open a browser and navigate to ***`localhost:5002`*** to start the producer event. This will start publishing data to the kafka broker network
- Subscribers are mapped from addresses ***6001- 6010***. (The number of subscribers can be scaled up further as well). Visit ***'localhost:6001'*** or any of the subscriber addresses to get to the subscriber UI
- Provide any username to log into the system. This ensures that you only get the messages you are supposed to get, and not another user's messages
- In the next screen, pick any topic from the dropdown to add a subscription.

- The unsubscribe tab can be used to unsubscribe from your list of subscriptions. If a topic has been unsubscribed the user will not see messages from that topic
- Switch to the notifications tab to view the messages from the subscribed topics. The notification page refreshes automatically every 10 seconds.
- To advertise a new topic, visit ***'localhost:5002/addtopics'*** to add a topic. This will create a floating marquee in the subscriber UI indicating the user of a new topic to subscribe to
- The user can now find this topic in the subscriber dropdown and can subscribe to this to see messages specific to this topic

7. Steps to run the code

Please find below the steps to be followed to obtain a successful implementation of Project 2

1) In Terminal, navigate to the kafka folder and run the following command
- `docker-compose up -d`
- This will create separate container for Zookeeper, Kafka message broker cluster and initiate it. This container will be placed under a docker network.

2) In another terminal, navigate to the api_call folder and run the following command
- `docker build -t apicall-image .`
- `docker run --name apicall-container -p 5002:5002 apicall-image`
- This will create a container designed for fetching publishers' data from the GitHub

3) Connect the API Data provider to the same docker network
- `docker network connect kafka_default apicall-container`

4) In another Terminal, navigate to frontend folder and run to create the frontend docker image
- `docker build -t frontend-image .`

5) Use the image to create the front-end container

- `docker-compose up -d`
- This creates ten instances of the front-end image. You can extend this and create multiple instances of front-end containers (Subscribers). These needs to be exposed to different local ports. Get the list of ports in which the front-end containers are running so that we could access it later. We consider 6003 as one of the ports.

6) To verify if all three containers are in the same network run,
- `docker network inspect kafka_default`

7) Open a browser and go to
- `localhost:5002`
- This will start the data fetcher container and publish/produce the appropriate GitHub events to the Kafka cluster
- You can add topics to the system by going to `localhost:5002/addtopics` and provide appropriate inputs.

8) Open a browser and go to
- `localhost:5003`
- Provide appropriate inputs. These inputs will be considered as a topic and polled for messages from Kafka cluster.
- This is extensible to all the containers created with the frontend-image

9) If you don't want any hassle running all the above commands, just `cd` to the project directory and run
- `./run-all.sh`

8. Team Contribution

Name	Person Number	Contribution
Shriram Ravi	50419944	a. Producer API code b. Producer dockerization c. Kafka broker setup d. Consumer polling
Siddharth Sankaran	50421657	a. Consumer API code b. Consumer dockerization c. Kafka subscription and unsubscription d. Multiple subscriber creation (docker-compose)